

Lecture 7 & 8:

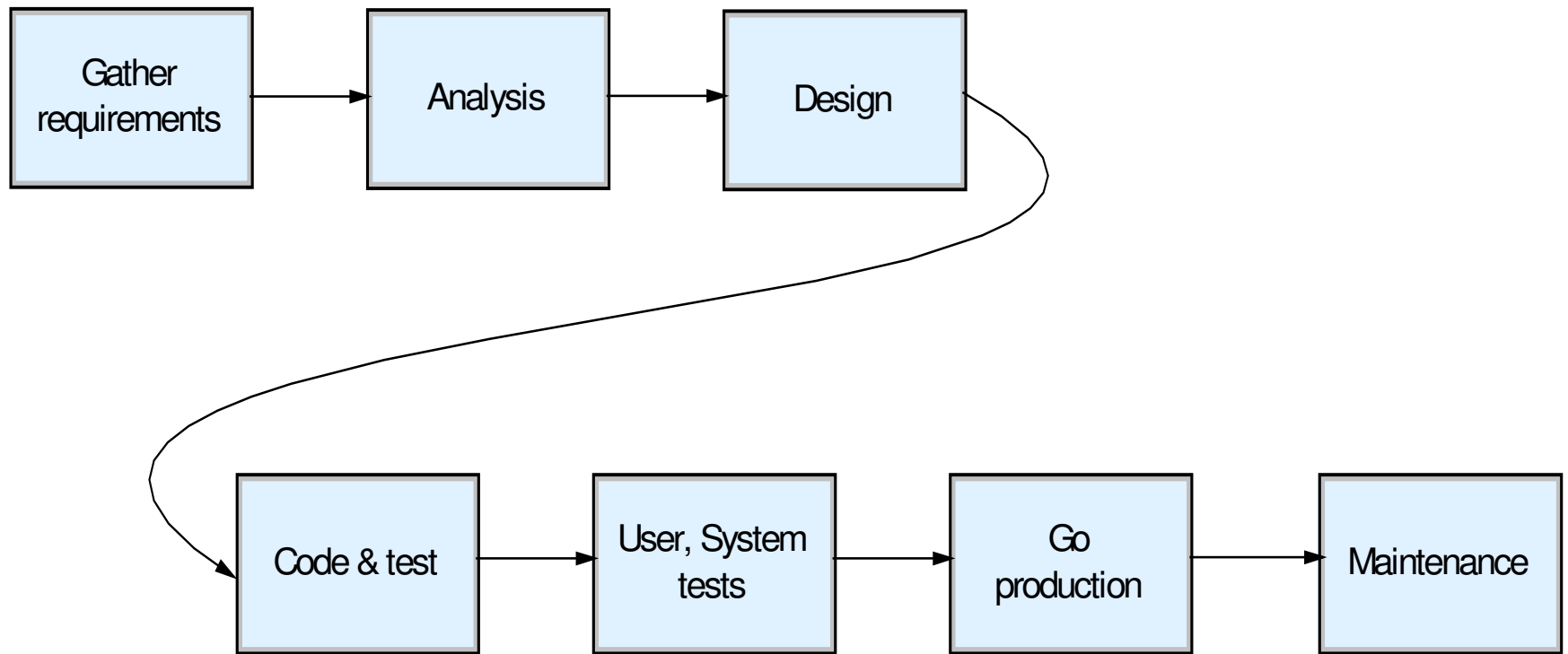
Creating Large Scale Applications

Reviewing the Software Development Life Cycle

Overview of Development Life Cycle

- Designing and developing an application for the mainframe is similar to other platforms, but some of the questions and conclusions are different.
- Life cycle of designing and developing an application to run on z/OS includes phases of:
 - Requirements gathering and analysis
 - Design
 - Development
 - Test and debugging
 - Production
 - Maintenance

Application Development Lifecycle



What Percentage of Time is Typically Spent “Coding”?

Analysis	Design	Build	Test

What Percentage of Time is Typically Spent “Coding”?

Analysis	Design	Build	Test
15%	25%	25%	35%

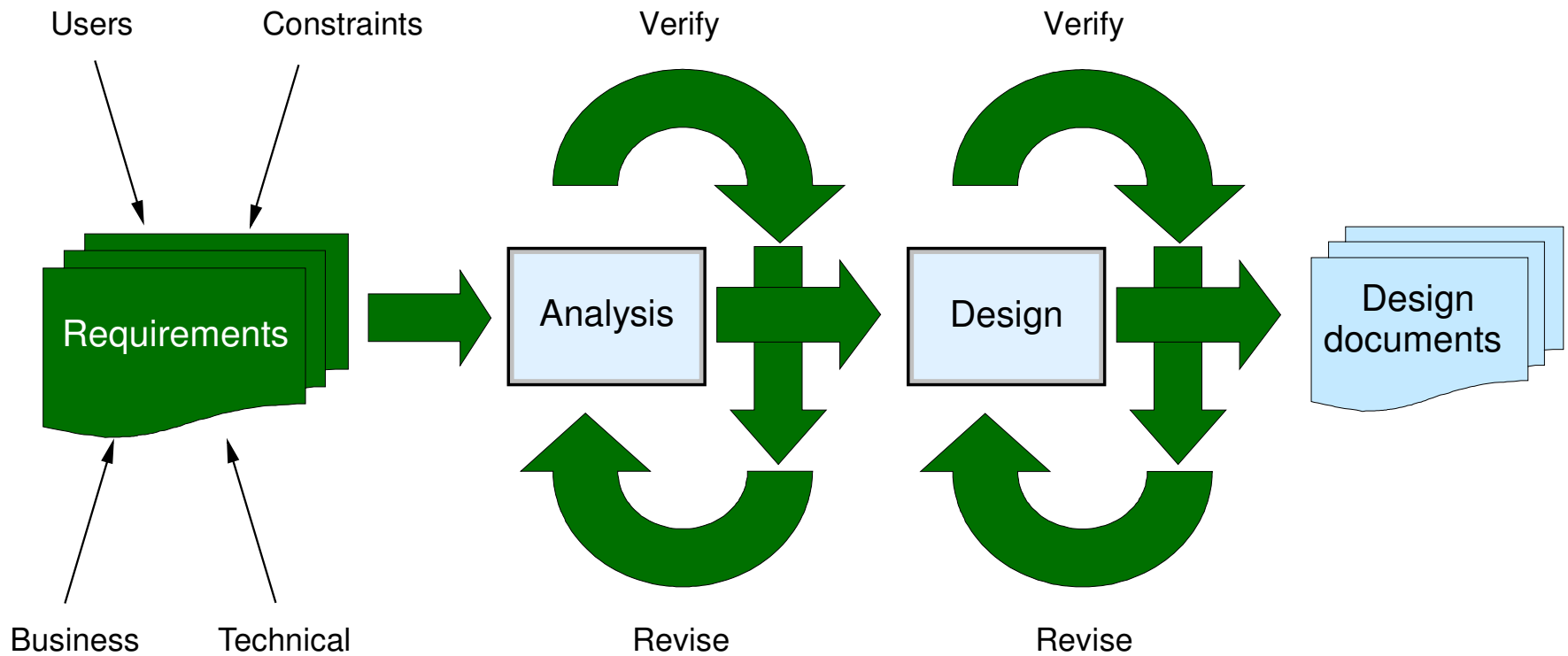
Gathering Requirements for the Design

- Requirements:
 - Assess what needs to be accomplished
 - Based on projects constraints
 - Always keep in mind the end result
 - Conduct interviews with users and stakeholders
 - State and verify our assumptions

Types of Requirements

- Accessibility
- Client
- Interoperability
- Recoverability
- Serviceability
- Availability
- Connectivity
- Performance
- Portability
- Usability
- Frequency of data backup
- Distributed
- Secure centralized controllable capacity
- Web services
- Changeability
- Preventing failure and fault analysis
- Resource can be monitored, controlled, managed, and administered

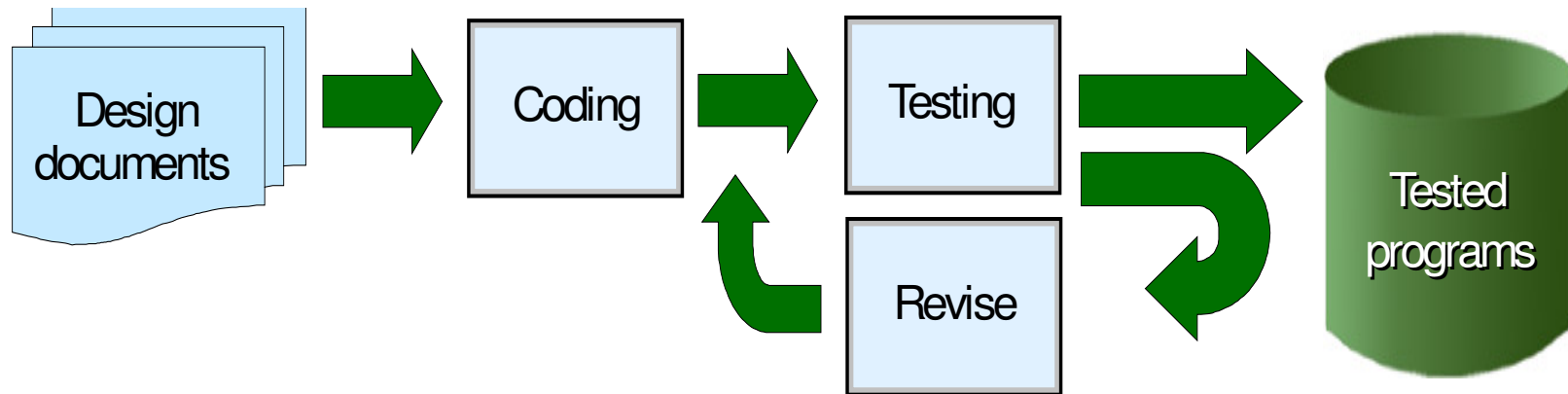
Design Phase



Design Decisions – *based on requirements*

- Batch versus online
- Database, tape, flat file, etc.
- Capacity of server
- Server type
- Develop or purchase package or both

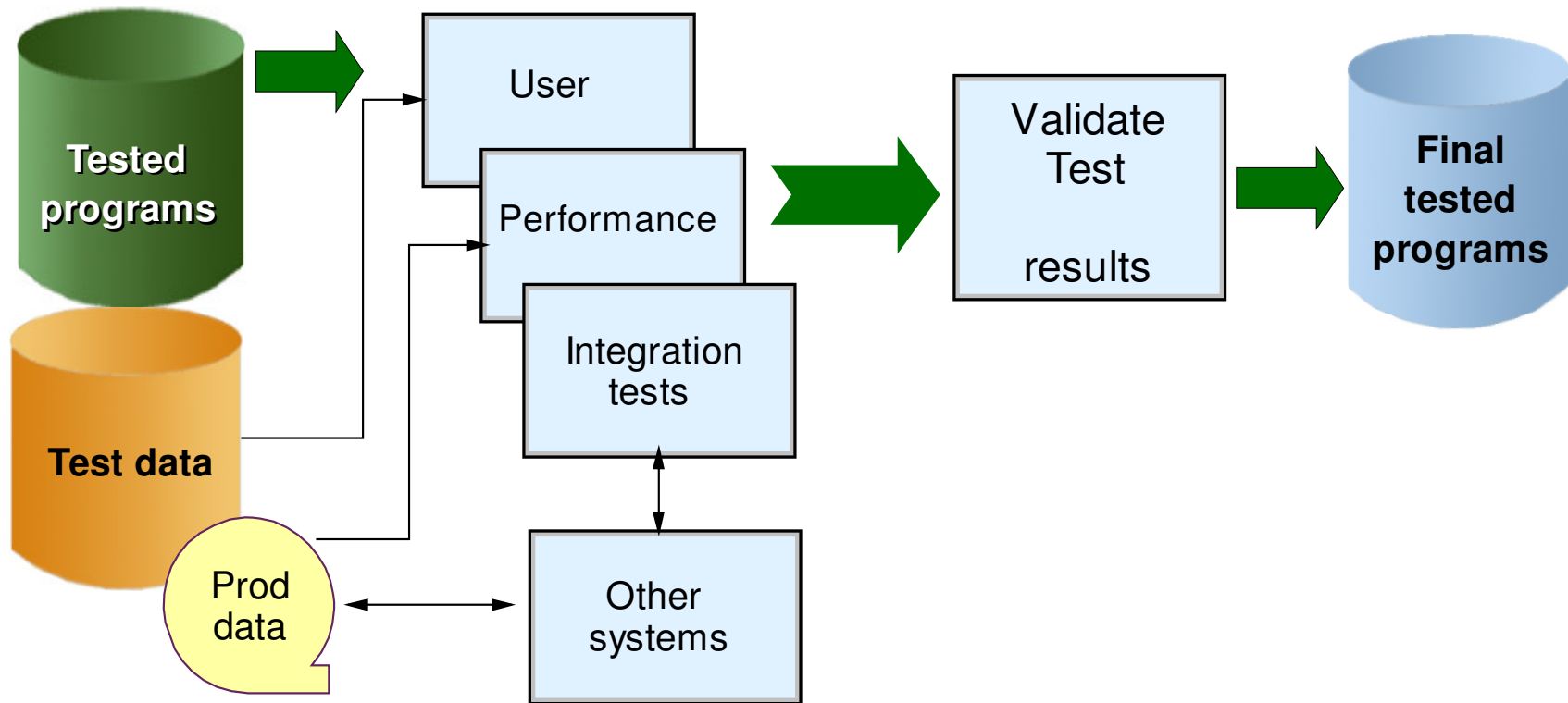
Development Phase



Developing an Application for the Mainframe

- Programmer uses as input the specifications of the designer
- Usually follows this process:
 - Code a module.
 - Test a module for functionality.
 - Make corrections to the module.
 - Repeat from step 2 until successful.

Test Phase



Test Phase Questions

- How to do performance testing if you can only afford one system, and it's in production?

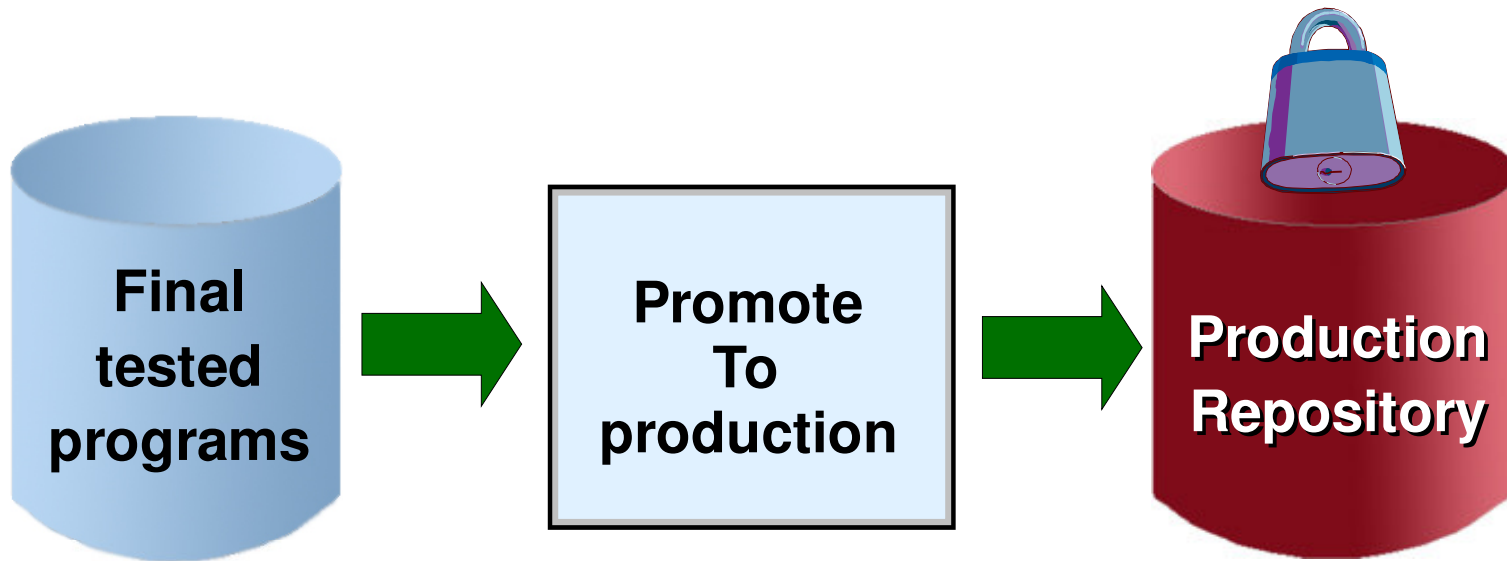
Test Phase Questions

- How to do performance testing if you can only afford one system, and it's in production?
- Why is it better to separate the role of the tester and the role of the developer

Test Phase (continued)

- Many levels of testing
 - User testing for functionality, acceptance
 - Performance (stress) testing
 - Integration testing (with other systems)
- Validate the testing results
- Final step before going production

Production Phase



Go Production

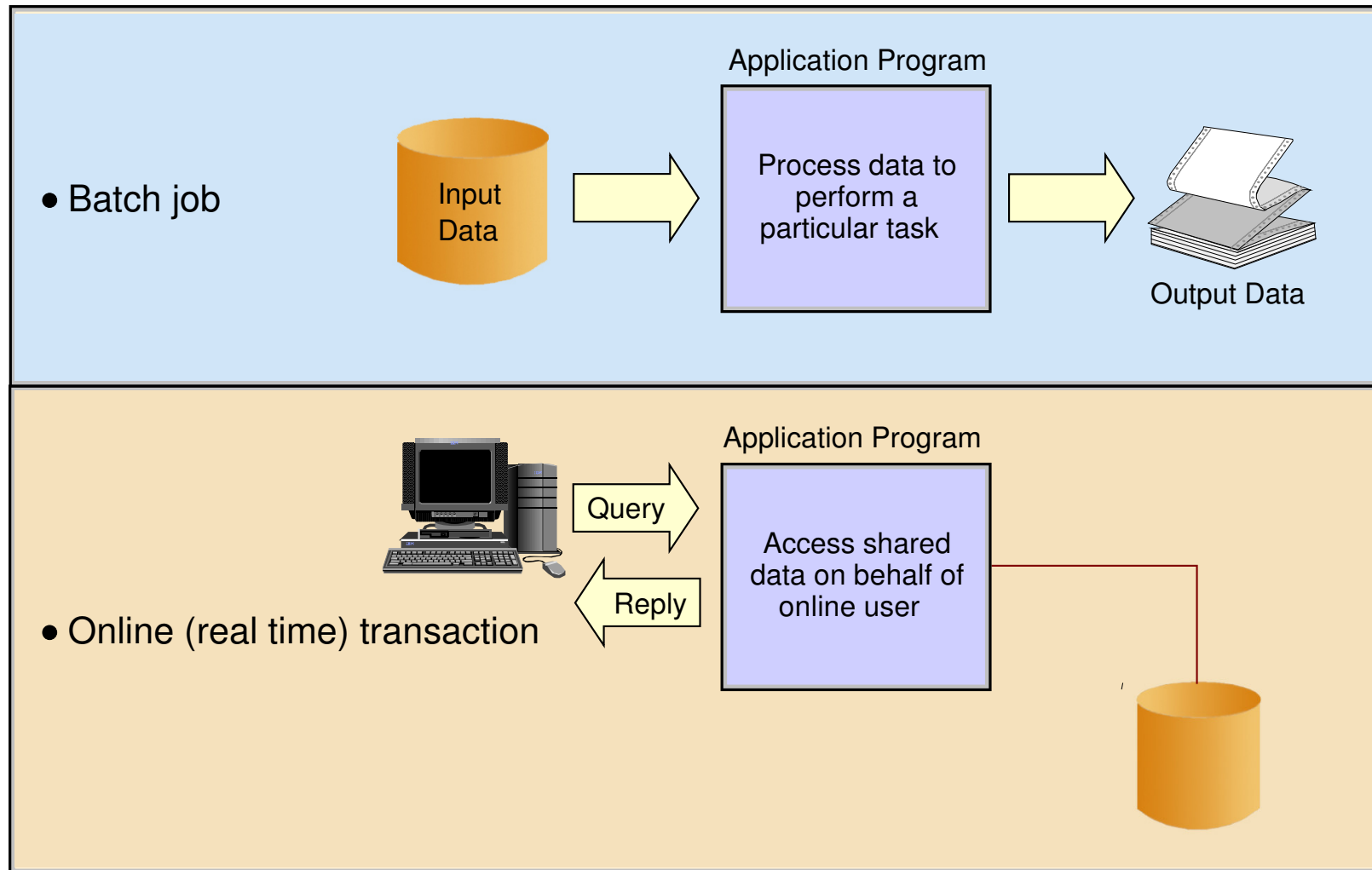
- Document:
 - Operational procedures
 - Training manuals (users, administrators, etc.)
- Promote application to production status
 - Implement change control process
- Hand over to operations

Maintenance Phase

- Ongoing day-to-day changes/enhancements
- Responsibility for maintenance may change to another group or stay with developers

Types of Applications: Batch & Online Processing

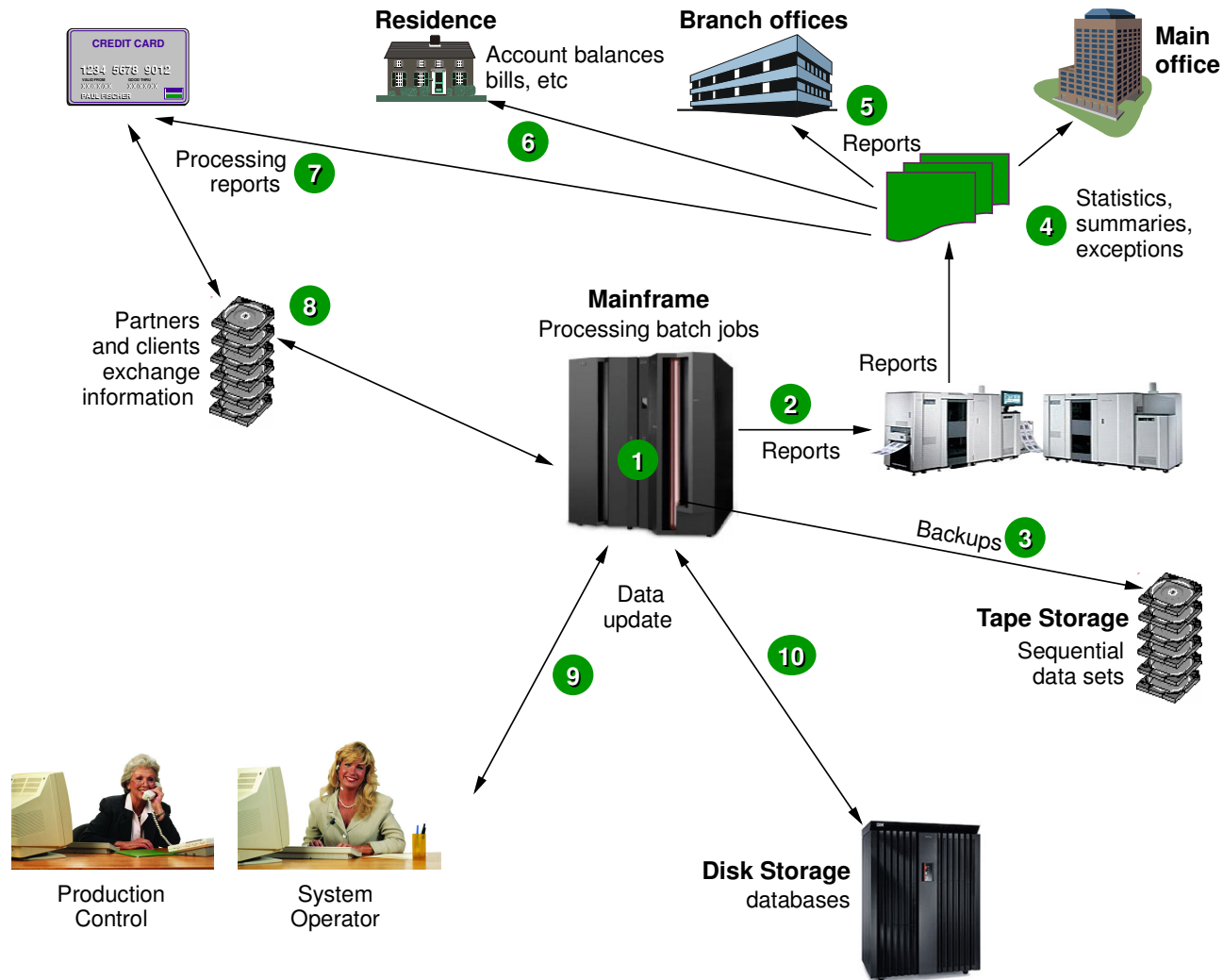
Typical mainframe workloads



What is batch processing?

- Batch processing is used for programs that can be executed:
 - With minimal human interaction
 - At a scheduled time or on an as-needed basis.
- After a batch job is submitted to the system for execution, there is normally no further human interaction with the job until it is complete.

Typical batch use



Examples of batch processing:

Example of batch processing

Examples of batch processing:

- New Credit Card Applications

Examples of batch processing:

- New Credit Card Applications
- Payments on credit cards

Examples of batch processing:

- New Credit Card Applications
- Payments on credit cards
- Credit Card Statements / Interest rate calculations

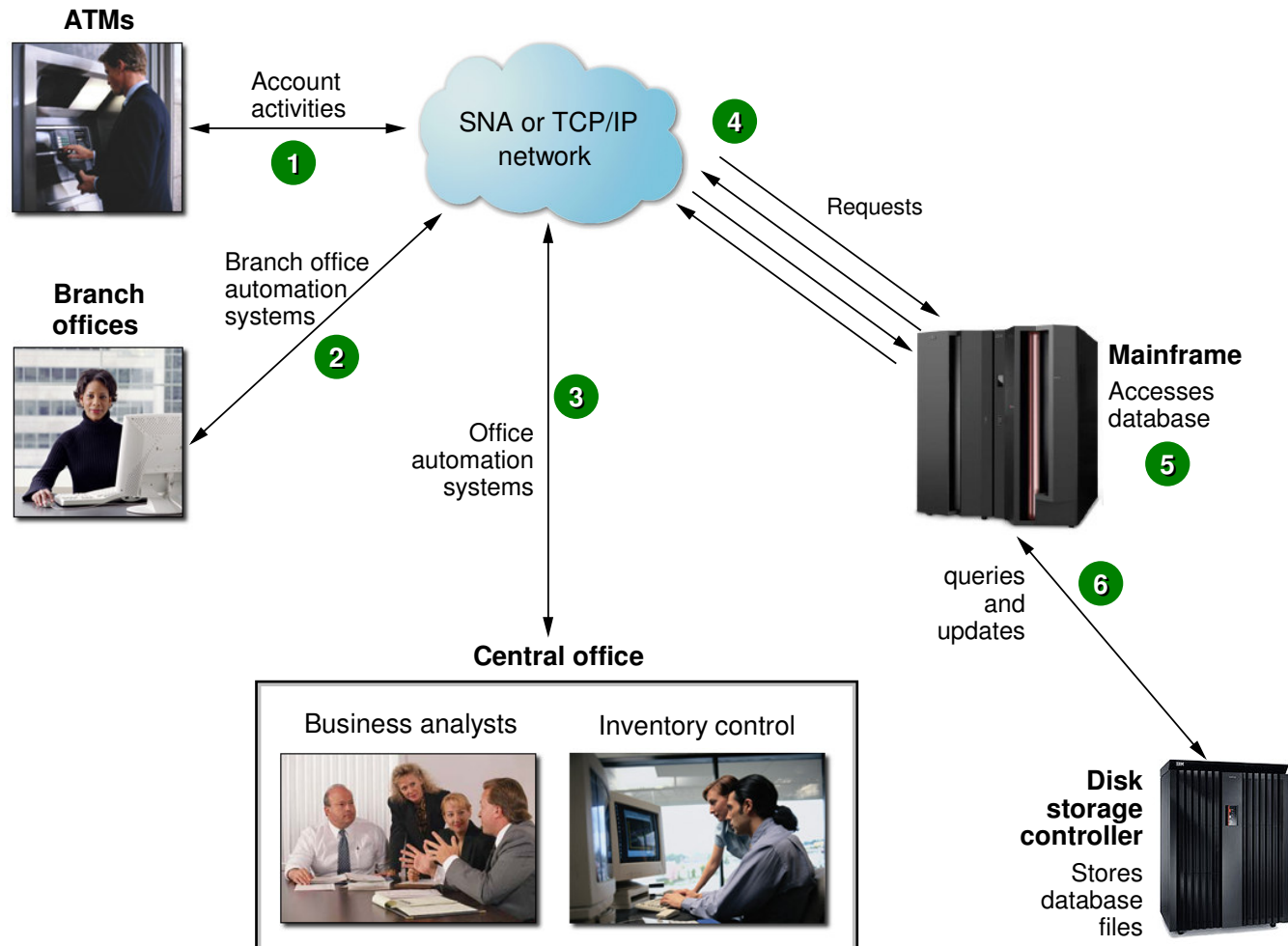
Example of online processing: a travel agency

- Mainframe applications designed for:
 - employee and customer information
 - contacts with car rental companies
 - hotels
 - airline schedules
- Changes must be immediately reflected to application end-users (in real time)
- Contrast with batch processing

What is online processing?

- Online processing is used for programs that require:
 - Human interaction
 - Can not be a scheduled

Typical online use



When to use which Batch vs Online Processing

- Reasons for using batch:
 - Data is stored on tape
 - Transactions are submitted for overnight processing
 - User does not require access to the data
- Reasons for using online:
 - Users require access to the data
 - Short response time required

COBOL (Part II)

How to use copy members

Copy member concepts

- A *copy member* contains COBOL source statements that can be copied into a COBOL source program.
- On a mainframe, a copy member is stored in a *partitioned data set* called a *copy library*, or *source statement library*.
- To identify the copy library when the program is compiled, you use JCL.

The basic syntax of the Copy statement

`COPY member-name`

An IBM mainframe COBOL example

`COPY CUSTMAST.`

A copy member that's named CUSTMAST

```
01  CUSTOMER-MASTER-RECORD.  
    05  CM-BRANCH-NUMBER          PIC 9(2) .  
    05  CM-SALESREP-NUMBER        PIC 9(2) .  
    05  CM-CUSTOMER-NUMBER        PIC 9(5) .  
    05  CM-CUSTOMER-NAME          PIC X(20) .  
    05  CM-SALES-THIS-YTD         PIC S9(5)V99 .  
    05  CM-SALES-LAST-YTD         PIC S9(5)V99 .
```

How to identify the copy library in the JCL that compiles the program

```
//COBOL.SYSLIB DD DSN=MM01.TEST.COPYLIB,DISP=SHR
```

How the member looks in the compiler listing

```
          COPY CUSTMAST.
C          01  CUSTOMER-MASTER-RECORD.
C          05  CM-BRANCH-NUMBER          PIC 9(2) .
C          05  CM-SALESREP-NUMBER        PIC 9(2) .
C          05  CM-CUSTOMER-NUMBER        PIC 9(5) .
C          05  CM-CUSTOMER-NAME          PIC X(20) .
C          05  CM-SALES-THIS-YTD          PIC S9(5)V99.
C          05  CM-SALES-LAST-YTD         PIC S9(5)V99.
```

How to create a copy member

1. Create a new member and enter the source code for it just as you would if you were writing a complete program.
2. Save the code as a member in a copy library.

The complete syntax of the Copy statement

COPY member-name [{OF | IN} library-name]

$$\left[\begin{array}{c} \text{REPLACING} \end{array} \left\{ \begin{array}{l} ==\text{pseudo-text-1}== \\ \text{identifier-1} \\ \text{literal-1} \\ \text{word-1} \end{array} \right\} \text{BY} \left\{ \begin{array}{l} ==\text{pseudo-text-2}== \\ \text{identifier-2} \\ \text{literal-2} \\ \text{word-2} \end{array} \right\} \dots \right]$$

Description

- On some platforms, you can use the In or Of clause to identify the library that the copy member is in.
- You can use the Replacing clause to find and replace text within a copy member as it is copied into your program.

A Copy statement that replaces all occurrences of the prefix CM with the prefix CMR and all 05 levels with 10s

```
COPY CUSTMAST  
  REPLACING ==CM-== BY ==CMR-==  
            ==05== BY ==10== .
```

COPY Example

Copyfile3.cbl

```
02  CustOrder          PIC 9(R) .
```

```
01 CopyData.  
COPY Copyfile3 REPLACING ==R== BY ==4==.
```

```
01 CopyData.  
    02  CustOrder          PIC 9(4) .
```

Typical copy members

- Record descriptions in the File Section or Working-Storage Section of the Data Division
- Data in the Working-Storage Section like the descriptions for the current time and date fields that are returned by the Current-Date function
- Table descriptions for tables that don't change like a state or zip-code table
- Processing routines that are used by more than one program like common calculating, printing, and editing routines

Recommended guidelines for using copy members

- Document all copy members so it's easy for programmers to find what they're looking for.
- Before you start a new program, find out whether any copy members are available that will help you develop the program more efficiently.
- Avoid using the Replacing clause.

How to use subprograms

CALL Example

**CALL "DateValidate"
USING BY CONTENT TempDate
USING BY REFERENCE DateCheckResult.**

**IDENTIFICATION DIVISION.
PROGRAM-ID DateValidate IS INITIAL.**

DATA DIVISION.

WORKING-STORAGE SECTION.

??????????????

LINKAGE SECTION.

01 DateParam

PIC X(8).

01 DateResult

PIC 9.

PROCEDURE DIVISION USING DateParam, DateResult.

Begin.

??????????????

??????????????

EXIT PROGRAM.

???????

??????????????

CALL Parameters

CALL "ProgramName" USING P1, P2, P3, P4.

PROCEDURE DIVISION USING P2, P4, P1, P3.

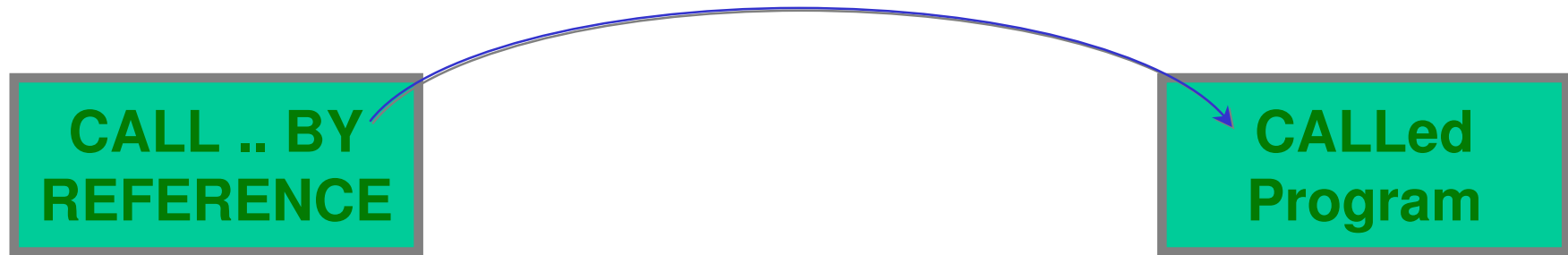
CALL Parameters

CALL "ProgramName" USING P1, P2, P3, P4.

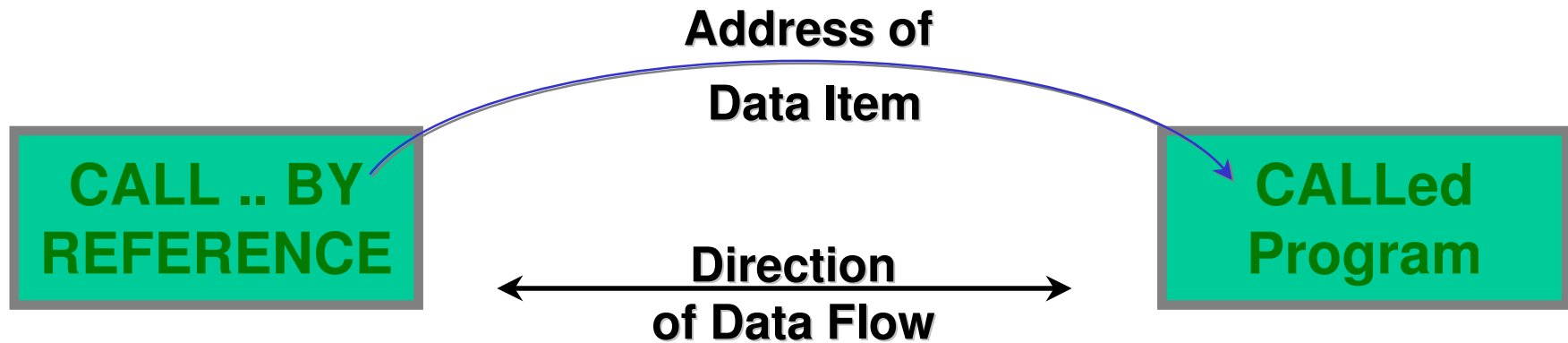
PROCEDURE DIVISION USING P2, P4, P1, P3.

Positions Correspond - Not Names

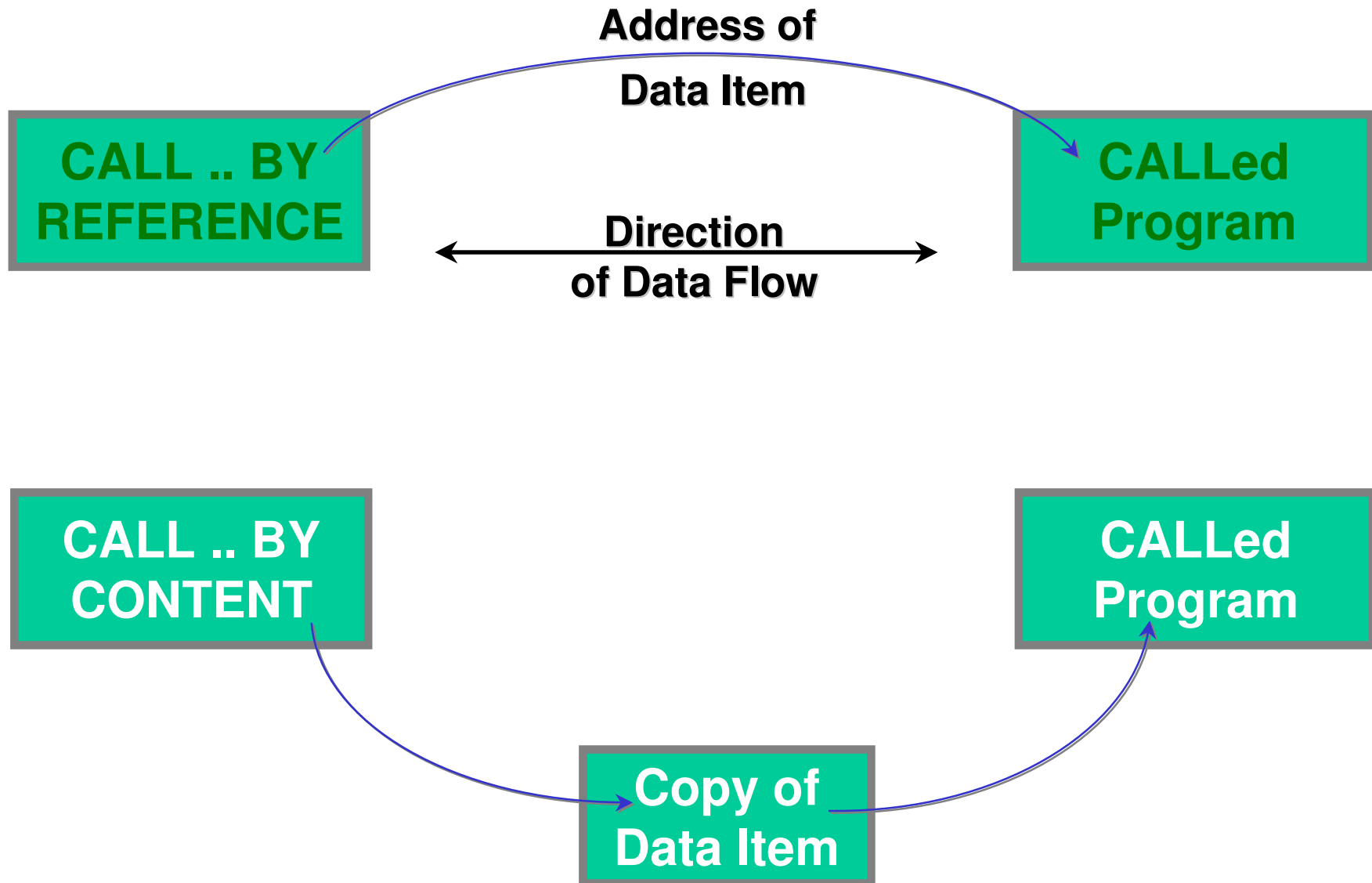
Parameter Passing Mechanisms



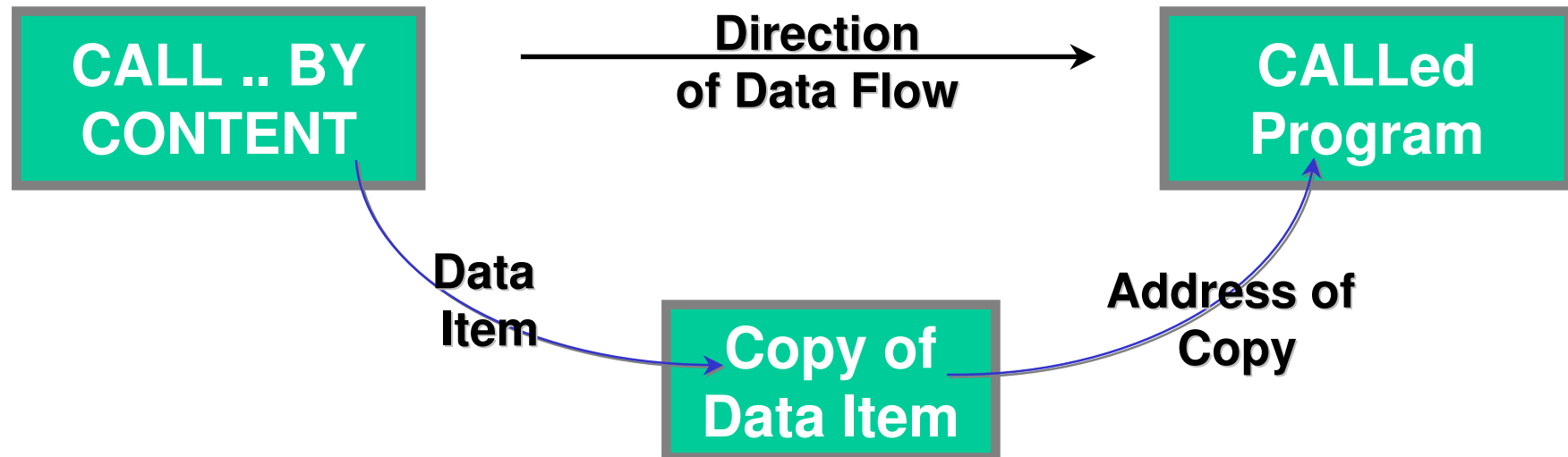
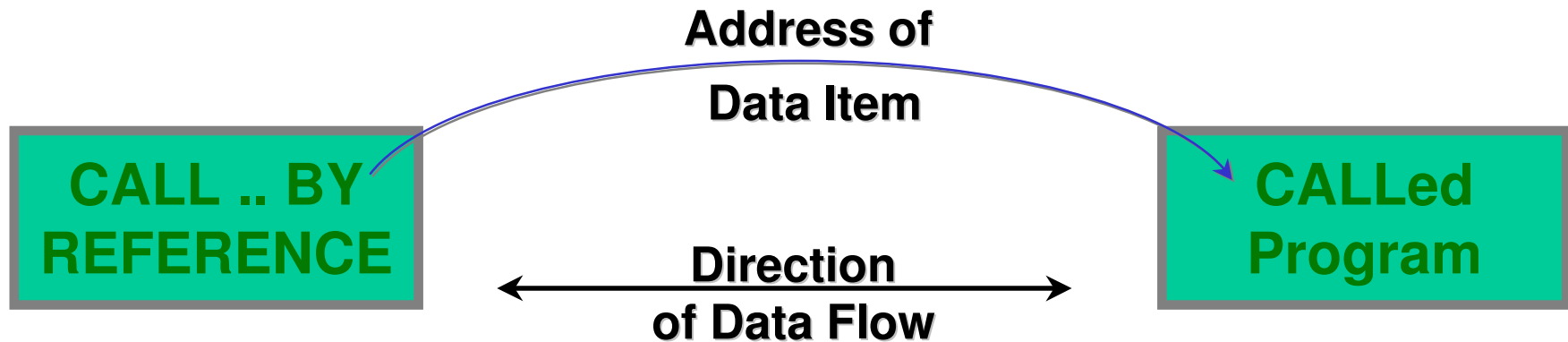
Parameter Passing Mechanisms



Parameter Passing Mechanisms



Parameter Passing Mechanisms



Contained Sub-Programs

IDENTIFICATION DIVISION.

PROGRAM-ID. MainProgram.

? ? ? ? ? ? ? ?

01 TableItem **IS GLOBAL**.

PROCEDURE DIVISION.

? ? ? ? ? ? ? ?

CALL PutToTable USING BY CONTENT DataItem

? ? ? ? ? ? ? ?

CALL ReportFromTable.

EXIT PROGRAM.

IDENTIFICATION DIVISION.

PROGRAM-ID. PutToTable.

? ? ? ? ? ? ? ?

END-PROGRAM PutToTable.

IDENTIFICATION DIVISION.

PROGRAM-ID. ReportFromTable.

? ? ? ? ? ? ? ?

END-PROGRAM ReportFromTable.

END-PROGRAM MainProgram.

Contained Sub-Programs

IDENTIFICATION DIVISION.

PROGRAM-ID. MainProgram.

? ? ? ? ? ? ? ?

01 TableItem **IS GLOBAL**.

PROCEDURE DIVISION.

? ? ? ? ? ? ? ?

CALL PutToTable USING BY CONTENT DataItem

? ? ? ? ? ? ? ?

CALL ReportFromTable.

EXIT PROGRAM.

IDENTIFICATION DIVISION.

PROGRAM-ID. PutToTable.

? ? ? ? ? ? ? ?

END-PROGRAM PutToTable.

IDENTIFICATION DIVISION.

PROGRAM-ID. ReportFromTable.

? ? ? ? ? ? ? ?

END-PROGRAM ReportFromTable.

END-PROGRAM MainProgram.

Contained Sub-Programs

IDENTIFICATION DIVISION.

PROGRAM-ID. MainProgram.

? ? ? ? ? ? ? ? ?

01 TableItem **IS GLOBAL**.

PROCEDURE DIVISION.

? ? ? ? ? ? ? ? ?

CALL PutToTable USING BY CONTENT DataItem

? ? ? ? ? ? ? ? ?

CALL ReportFromTable.

EXIT PROGRAM.

IDENTIFICATION DIVISION.

PROGRAM-ID. PutToTable.

? ? ? ? ? ? ? ? ?

END-PROGRAM PutToTable.

IDENTIFICATION DIVISION.

PROGRAM-ID. ReportFromTable.

? ? ? ? ? ? ? ? ?

END-PROGRAM ReportFromTable.

END-PROGRAM MainProgram.

The basic syntax of the Call statement

```
CALL "subprogram-name" [USING identifier-1 ...]
```

A Call statement with four parameters

```
CALL "CALCFV" USING INVESTMENT-AMOUNT  
                     NUMBER-OF-YEARS  
                     YEARLY-INTEREST-RATE  
                     FUTURE-VALUE.
```

The data definitions for the fields passed by the calling program

```
01  USER-ENTRIES.  
    05  INVESTMENT-AMOUNT          PIC 99999.  
    05  NUMBER-OF-YEARS           PIC 99.  
    05  YEARLY-INTEREST-RATE      PIC 99V9.  
01  WORK-FIELDS.  
    05  FUTURE-VALUE              PIC 9(7)V99.
```

How to identify the subprogram library in the JCL that compiles the program

```
//LKED.SYSLIB DD  
//           DD DSN=MM01.TEST.OBJLIB,DISP=SHR
```

How to call a subprogram

- A *subprogram* is a program that is *called* by a *calling program*.
- Before a subprogram can be called by another program, it must be compiled and ready for execution.
- To call a subprogram, you code the name of the subprogram, or *object module* in a Call statement.
- The Using clause identifies the fields in the calling program that are *passed* to the subprogram. These can be referred to as *parameters* or *arguments*.
- The Using clause must list the passed fields in the sequence that the subprogram requires.
- When you compile and run the calling program, you use JCL to identify the *object library* that the subprogram is stored in. Then, the calling program and the subprogram are *link-edited* into a *load module* that can be run by the system.

The syntax that for creating a subprogram

LINKAGE SECTION.

(Data definitions for fields that are passed by the calling program.)

PROCEDURE DIVISION USING identifier-1 ...

.

.

EXIT PROGRAM.

Description

- The Linkage Section defines the fields that must be passed to the subprogram from the calling program.
- The Using clause in the Procedure Division header lists the fields that must be passed to the subprogram.
- The calling program must define the passed fields the same way they're defined in the Linkage Section.
- The Call statement must list the passed fields in the same sequence that they're listed in the Procedure Division statement.

Testing and debugging considerations for subprograms

- If an existing subprogram doesn't work correctly, it's usually because the calling program has passed the parameters in a different sequence or format than the subprogram requires.
- You can write a simple calling program to test your subprogram if none of the programs that use it have been developed yet.

Recommended guidelines for using subprograms

- Write each subprogram so it does a single function and has only one entry and one exit point.
- Document all subprograms so they're easy to find.
- Before you start a new program, find out whether you can use any existing subprograms.
- When you design a new program, include a module for each subprogram in the structure chart.

Using Copybooks and subroutines

```
//MYACCTM JOB
//STEP1 EXEC
PROC=IGYWC,PARM.COBOLE='LIB,OBJECT,XREF,FLAG(I,E)'
//COBOLE.SYSIN DD DSN=KCxx.MINI.COBOLE(ACCT),DISP=SHR
//*
//* DEFINE WHERE TO FIND THE COPYBOOKS
//COBOLE.SYSLIB DD DSN=KC02170.MINI.COPYLIB,DISP=SHR
//*
//* DEFINE WHERE THE OBJECT WILL GO
//COBOLE.SYSLIN DD DSN=KCxx.MINI.OBJLIB(ACCT),DISP=SHR
```

Compiling a source program and storing in the objlib for later linking...

Using Copybooks and subroutines

Linking with other compiled modules in objlib

```
//MYMAIN JOB
//STEP1 EXEC PROC=IGYWCLG,PARM.COBOLE='LIB,OBJECT,XREF,FLAG(I,E)'
//COBOLE.SYSIN DD DSN=KC02170.MINI.COBOLE(MAIN),DISP=SHR
//*
//* DEFINE WHERE TO FIND THE COPYBOOKS
//COBOLE.SYSLIB DD DSN=KC02170.MINI.COPYLIB,DISP=SHR
//*
//* WHERE TO PUT OUTPUT EXECUTABLE
//LKED.SYSLMOD DD DSN=KC02170.MINI.LOAD(MAIN),DISP=SHR
//*
//* DEFINE WHERE THE OTHER OBJECT FILES ARE LOCATED
//LKED.SYSLIB DD
//      DD DSN=KC02170.MINI.OBJLIB,DISP=SHR
//*
//* DEFINE THE INPUT/OUTPUT DATASET
//GO.NXTACCN DD DSN=KC02170.DATA.NXTACCN,DISP=SHR
//GO.ACCTMAST DD DSN='KC02170.DATATEST.T08',DISP=SHR
//GO.NEWACCTS DD DSN=KC02170.DATA.NEWACCTS,DISP=SHR
//GO.SYSOUT DD SYSOUT=*
```

Sort & Merge

Sorting

- The StudentFile is a sequential file sequenced upon ascending StudentId.
- Write a program to display the number of students taking each course. How?

```
DATA DIVISION.  
FILE SECTION.  
FD StudentFile.  
01 StudentDetails.  
    02 StudentId          PIC 9(7) .  
    02 StudentName.  
        03 Surname        PIC X(8) .  
        03 Initials       PIC XX .  
    02 DateOfBirth.  
        03 YOBirth        PIC 9(2) .  
        03 MOBirth        PIC 9(2) .  
        03 DOBirth        PIC 9(2) .  
    02 CourseCode        PIC X(4) .  
    02 Grant              PIC 9(4) .  
    02 Gender             PIC X .
```

Sorting - using COBOL

```
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.
```

```
    SELECT WorkFile ASSIGN TO "WORK.TMP".
```

```
SD  WorkFile.  
01  WorkRecord.  
    02  ProvinceCode          PIC 9.  
    02  SalesmanCode          PIC 9(5) .  
    02  FILLER                PIC X(19) .
```

```
PROCEDURE DIVISION.  
Begin.
```

```
    SORT WorkFile ON ASCENDING KEY ProvinceCode  
                  DESCENDING KEY SalesmanCode  
    USING UnsortedSales  
    GIVING SortedSales.
```

```
    OPEN INPUT SortedSales.
```

Sorting - using JCL

- But most people do not use Cobol **internal** sorting because you can also sort using JCL

```
//JOBSORT JOB
//STEP1 EXEC PGM=ICETOOL
//IN          DD DSN='KCXXXX.LAB2.NEWACCTS',DISP=SHR
//SYSOUT      DD *
//TOOLMSG     DD SYSOUT=*
//DFSMSG      DD SYSOUT=*
//OUT         DD DSN=KCXXXX.LAB2.NEWSORT,DISP=SHR
//TOOLIN      DD *
              SELECT FROM(IN) TO(OUT) ON(1,10,CH) LAST
/*
//*
```

For duplicates, can have: LAST, FIRST, NODUPS, ALLDUPS

MERGE Description

- The Merge takes two or more identically sequenced files and combines them, according to the key values specified, to produce a combined file which is then output to an output file or OUTPUT PROCEDURE.
- This can also be done in COBOL or JCL

e.g.

```
MERGE WorkFile ON ASCENDING KEY StudentId  
      USING InsertionsFile, StudentFile  
      GIVING NewStudentFile.
```